МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья.

Студент гр. 8304 _____ Бутко А.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Изучить бинарные деревья и леса, реализовать бинарное дерево на векторе.

Текст задания 4-В.

Заданы два бинарных дерева b1 и b2 типа BT с произвольным типом элементов. Проверить:
- подобны ли они (два бинарных дерева подобны, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);
- равны ли они (два бинарных дерева равны, если они подобны и их соответствующие элементы равны);
- зеркально подобны ли они (два бинарных дерева зеркально подобны, если они оба пусты либо они оба непусты и для каждого из них левое поддерево одного подобно правому поддереву другого);
- симметричны ли они (два бинарных дерева симметричны, если они зеркально подобны и их соответствующие элементы равны).

Описание алгоритма.

Программа считывает две строки, после чего создается лес из двух бинарных деревьев.

Сравнение и выявление подобия осуществляется с помощью двух рекурсивных функций, одна из которых сравнивает два дерева «прямым» способом, а другая «зеркальным». В «прямом» способе сравниваются значения самих элементов и значения индексов сыновьих элементов сначала по левую сторону от элемента-родителя первого и второго деревьев, затем по правую сторону. Каждый раз сравнивая значения переходим к сыновьям элементам и рекурсивно запускаем функцию, пока не дойдем до конечных элементов хотя бы одного дерева. В «зеркальном» способе действуем аналогично, однако сравниваем значения самих элементов и значения индексов левого сыновьего элемента одного дерева и правого сыновьего элемента другого дерева и наоборот.

Описание функций программы.

1. `explicit BinaryTree(int size)`

Конструктор класса BinaryTree, который принимает размер начального массива.

2. `bool makeBinaryTree(int& index, const std::string& str)`

Функция создания бинарного дерева из строки, введенной пользователем.

3. `void resize()`

Функция, предназначенная для выделения дополнительной памяти, если не хватит выделенной изначально.

4. `void print(Element* root, int I = 0)`

Рекурсивная функция печати бинарного дерева по глубине, I - глубина.

5. `void setLeftIndex(int currentIndex, int leftIndex = -1)`

Функция предназначена для установки индекса сыновьего элемента текущему элементу (аналогичная функция `setRightIndex`).

6. `void setElementValue(T value, int index)`

Функция присваивания значения элементу массива.

7. `BinaryForest(int size1, int size2) : tree1(size1), tree2(size2)`

Конструктор класса BinaryForest, который принимает размеры начальных массивов для двух деревьев.

8. `bool makeBinaryForest(const std::string &str1,const std::string &str2, int& index1, int& index2)`

Функция, которая дважды вызывает функцию под пунктом 2 для создания двух бинарных деревьев.

9. `void printBinaryForest()`

Функция, которая дважды вызывает функцию под пунктом 4 для печати двух бинарных деревьев.

10. `void forestComparison()`

Функция сравнения двух заданных деревьев, которая вызывает функции под пунктом 11 и 12.

11. `void forestDirectComparison(typename BinaryTree<T>::Element *root1, typename BinaryTree<T>::Element* root2)`

Рекурсивная функция «прямого» сравнения двух деревьев.

12. `void forestMirrorComparison(typename BinaryTree<T>::Element *root1, typename BinaryTree<T>::Element* root2)`

Рекурсивная функция «зеркального» сравнения двух деревьев.

Тестирование.

Входные данные	Выходные данные
(a(b)(b)) (d(e)(f))	Binary trees: similar; not equal; mirror similar; not mirror equal.
(e(r(t(r)(r))(e)) (a(b	ERROR: incorrect string
(a(b(c(d)(e))(r))(f)) (a(f)(b(r)(c(e)(d))))	Binary trees: not similar; not equal; mirror similar; mirror equal.
(a(b(c(d)(e))(r))(f)) (1(2(3(4)(5))(6))(7))	Binary trees: similar; not equal; not mirror similar; not mirror equal.
¯_(ツ)_/¯ так и живем	ERROR: incorrect string
(a(a)(a)) (a(a)(a))	Binary trees: similar; equal; mirror similar; mirror equal.
Qwerty qwqwrwr(fqwfqwqwrqwr	ERROR: incorrect string

Вывод.

Для решения данной задачи было не очень целесообразно использовать векторную реализацию, так как в случае с бинарными деревьями она лишь ухудшает понимание и, вероятно, практически не имеет никаких преимуществ над ссылочной реализацией.

Приложение А.

Файл main.cpp

```
#include "BinaryTree.h"

int main(int argc, char* argv[]) {
    int index1 = 0,
        index2 = 0,
        testCounter = 0;
    std::string str1,
                str2;
    if(argc == 1)
    {
        std::cout << "Input first string:" << std::endl;
        std::getline(std::cin, str1);
        std::cout << "Input second string:" << std::endl;
        std::getline(std::cin, str2);
    }
    else
    {
        std::cout << "For file: " << argv[1] << std::endl;
        std::ifstream inputFile(argv[1]);
        if (!inputFile.is_open())
        {
            std::cout << "ERROR: file isn't open" << std::endl;
            return 0;
        }
        if (inputFile.eof())
        {
            std::cout << "ERROR: file is empty" << std::endl;
            return 0;
        }
        while(std::getline(inputFile, str1) && std::getline(inputFile,
str2))
        {
            std::cout << std::endl << "Test №" << ++testCounter <<
std::endl;
```

```
            std::cout << "First string:" << std::endl;
            std::cout << str1 << std::endl;
            std::cout << "Second string:" << std::endl;
            std::cout << str2 << std::endl;
            BinaryForest<std::string> forest(str1.size(), str2.size());
            if (forest.makeBinaryForest(str1, str2, index1, index2)){
                forest.printBinaryForest();
                forest.forestComparison();
            }
            index1 = 0;
            index2 = 0;
        }
        return 0;
    }
    BinaryForest<std::string> forest(str1.size(), str2.size());
    if (forest.makeBinaryForest(str1, str2, index1, index2)){
        forest.printBinaryForest();
        forest.forestComparison();
    }
    return 0;
}
```

Приложение Б.

Файл BinaryTree.cpp

```
#pragma once

#include <iostream>
#include <string>
#include <array>
#include <fstream>

template <typename T>
class BinaryTree
{
public:
    struct Element
    {
        T value;
        size_t leftElementIndex,
               rightElementIndex;
    };
    Element** array;

    explicit BinaryTree(int size)
    {
        maxSize_ = size;
        array = new Element* [size];
        for (int i = 0; i < size; i++)
            array[i] = new Element;
    }

    void setElementValue(T value, int index)
    {
        array[index]->value = value;
    }

    void setLeftIndex(int currentIndex, int leftIndex = -1)
    {
        array[currentIndex]->leftElementIndex = leftIndex;
    }

    void setRightIndex(int currentIndex, int rightIndex = -1)
    {
        array[currentIndex]->rightElementIndex = rightIndex;
    }

    bool makeBinaryTree(int& index, const std::string& str)
    {
        int currentIndex = counter_;
        if (str[index] != '(')
        {
            std::cout<<"ERROR: incorrect string"<<std::endl;
            return false;
        }
```

```
        std::string tmpStr;
        while(str[++index] != '(' && str[index] != ')' && str[index] !=
'#' && index != str.size())
            tmpStr += str[index];
        if (tmpStr.empty() && str[index] == '(')
        {
            std::cout<<"ERROR: incorrect string"<<std::endl;
            return false;
        }
        setElementValue(tmpStr, currentIndex);
        if (str[index] ==')')
        {
            setLeftIndex(currentIndex);
            setRightIndex(currentIndex);
            ++index;
            return true;
        }
        if (str[index] == '#' && str[index + 1] != '(')
        {
            std::cout<<"ERROR: incorrect string"<<std::endl;
            return false;
        }
        else if (str[index] == '#')
        {
            setLeftIndex(currentIndex);
            ++index;
        }
        else if(str[index] == '(')
        {
            if(counter_ + 1 == maxSize_) resize();
            setLeftIndex(currentIndex, ++counter_);
            if(!makeBinaryTree(index, str)) return false;
        }
        if(str[index] == ')')
        {
            setRightIndex(currentIndex);
            ++index;
            return true;
        }
        else if(str[index] == '(')
        {
            if(counter_ + 1 == maxSize_) resize();
            setRightIndex(currentIndex, ++counter_);
            if(!makeBinaryTree(index, str)) return false;
        }
        else {
            std::cout << "ERROR: incorrect string" << std::endl;
            return false;
        }
        ++index;
        if(str[index] == ' ')
        {
            std::cout << "ERROR: incorrect string" << std::endl;
            return false;
```

```
        }
        return true;
    }

    void resize()
    {
        auto** tmpArray = new Element* [maxSize_*2];
        for (int i = 0; i < counter_; ++i)
            tmpArray[i] = array[i];
        maxSize_ = maxSize_*2;
        for (int i = 1 + counter_; i < maxSize_; ++i)
            tmpArray[i] = new Element;
        delete[] array;
        array = tmpArray;
    }

    void print(Element* root, int I = 0)
    {
        if (root->rightElementIndex != -1)
        {
            print(array[root->rightElementIndex], 1+I);
        }
        for (int i = 0; i < I; i++)
            std::cout << "    ";
        std::cout << root->value << std::endl;
        if (root->leftElementIndex != -1)
        {
            print(array[root->leftElementIndex], 1+I);
        }
    }

    ~BinaryTree()
    {
        for (int i = 0; i < maxSize_; ++i)
            delete(array[i]);
        delete[] array;
    }
private:
    int counter_ = 0;
    int maxSize_ = 20;
};

template <typename T>
class BinaryForest
{
public:
    BinaryTree<T> tree1;
    BinaryTree<T> tree2;
    bool isDirectResemblance = true;
    bool isDirectEquality = true;
    bool isMirrorResemblance = true;
    bool isMirrorEquality = true;

    BinaryForest(int size1, int size2) : tree1(size1), tree2(size2){}
```

```
    bool makeBinaryForest(const std::string &str1,const std::string
&str2, int& index1, int& index2)
    {
        return !(!tree1.makeBinaryTree(index1, str1) || !tree2.make-
BinaryTree(index2, str2));
    }

    void printBinaryForest()
    {
        std::cout << "_________________" << std::endl;
        std::cout << "Binary Tree №1 :" << std::endl;
        std::cout << "_________________" << std::endl;
        tree1.print(tree1.array[0]);
        std::cout << "_________________" << std::endl;
        std::cout << "Binary Tree №2 :" << std::endl;
        std::cout << "_________________" << std::endl;
        tree2.print(tree2.array[0]);
        std::cout << "_________________" << std::endl;
    }

    void forestComparison()
    {
        forestDirectComparison(tree1.array[0], tree2.array[0]);
        forestMirrorComparison(tree1.array[0], tree2.array[0]);
        std::cout << std::endl << "Binary trees:" << std::endl;
        if (isDirectResemblance) std::cout << "similar;" << std::endl;
        else std::cout << "not similar;" << std::endl;
        if (isDirectEquality) std::cout << "equal;" << std::endl;
        else std::cout << "not equal;" << std::endl;
        if (isMirrorResemblance) std::cout << "mirror similar;" <<
std::endl;
        else std::cout << "not mirror similar;" << std::endl;
        if (isMirrorEquality) std::cout << "mirror equal." << std::endl;
        else std::cout << "not mirror equal." << std::endl;
    }

    void forestDirectComparison(typename BinaryTree<T>::Element *root1,
typename BinaryTree<T>::Element* root2)
    {
        if (root1->value != root2->value)
            isDirectEquality = false;

        if (root1->rightElementIndex != -1 && root2->rightElementIndex !
= -1)
            forestDirectComparison(tree1.array[root1-
>rightElementIndex], tree2.array[root2->rightElementIndex]);
        else if (root1->rightElementIndex != -1 || root2->rightElement-
Index != -1)
        {
            isDirectEquality = false;
            isDirectResemblance = false;
        }
```

```
        if (root1->leftElementIndex != -1 && root2->leftElementIndex !=
-1)
            forestDirectComparison(tree1.array[root1->leftElementIndex],
tree2.array[root2->leftElementIndex]);
        else if (root1->leftElementIndex != -1 || root2->leftElement-
Index != -1)
        {
            isDirectEquality = false;
            isDirectResemblance = false;
        }
    }

    void forestMirrorComparison(typename BinaryTree<T>::Element *root1,
typename BinaryTree<T>::Element* root2)
    {
        if (root1->value != root2->value)
            isMirrorEquality = false;

        if (root1->rightElementIndex != -1 && root2->leftElementIndex !=
-1)
            forestMirrorComparison(tree1.array[root1-
>rightElementIndex], tree2.array[root2->leftElementIndex]);
        else if (root1->rightElementIndex != -1 || root2->leftElement-
Index != -1)
        {
            isMirrorEquality = false;
            isMirrorResemblance = false;
        }

        if (root1->leftElementIndex != -1 && root2->rightElementIndex !=
-1)
            forestMirrorComparison(tree1.array[root1->leftElementIndex],
tree2.array[root2->rightElementIndex]);
        else if (root1->leftElementIndex != -1 || root2->rightElement-
Index != -1)
        {
            isMirrorEquality = false;
            isMirrorResemblance = false;
        }
    }

    ~BinaryForest() = default;
};
```